Parte 1:

Aprendendo a trabalhar com sinais e a interagir com sistemas

Esta prática tem por propósito ilustrar a geração de sinais básicos e a trabalhar com eles com as principais funções do Matlab.

- **Ponto 1: geração de sinais básicos**

Amostra unitária $\delta(n)$: esta função é gerada usando o código impseq(n0, inicio, fim) conforme exemplo:

```
[sinal1, n1] = impseq(0, -3, +3)
sinal1 =
     0     0     0     1     0     0     0
n1 =
    -3    -2    -1     0     1     2     3

[sinal2, n2] = impseq(3, 1, 5)
sinal2 =
     0     0     1     0     0
n2 =
     1     2     3     4     5
```

Degrau unitário u(n): esta função é gerada usando o código stepseq(n0, inicio, fim) conforme exemplo:

```
[sinal1, n1] = stepseq(0, -3, +3)
sinal1 =
     0     0     0     1     1     1     1
n1 =
    -3    -2    -1     0     1     2     3

[sinal2, n2] = stepseq (3, 1, 5)
sinal2 =
     0     0     1     1     1
n2 =
     1     2     3     4     5
```

Sequência exponencial complexa: para entender melhor o comportamento de sinais, experimente gerar um e visualizá-lo conforme exemplo abaixo da função

```
n=[-5:30];
a=exp((0.2+3j)*n);
subplot(2,2,1); stem(n, real(a));  title('Parte real');
subplot(2,2,2); stem(n, imag(a));  title('Parte imaginária');
subplot(2,2,3); stem(n, abs(a));   title('Magnitude');
subplot(2,2,4); stem(n, angle(a)); title('Fase');
b=exp(0.2*n);
c=exp(3j*n);
figure;
subplot(2,1,1); stem(n, b); title('Módulo');
subplot(2,1,2); stem(n, c); title('Fase');
```

- **Ponto 2: Amostrando sinais**

Para simular o processo de conversão de tempo contínuo para tempo discreto, utilize o código abaixo.

```
%% inicializacoes
F = 60;          %freq analogica
Fs = 400;        %freq aquisicao
ts = 1/Fs;       %tempo aquisicao
num_ciclos = 3;  %(equivale a 3 ciclos)
t = 0:1e-6:(num_ciclos*(1/F)); %tempo simulacao
n = 0:round(num_ciclos*(Fs/F));        %numero amostras

%% parte analogica
f_analogico = sin(2*pi*F*t);
plot(t,f_analogico);

%% parte digital
f_digital = sin(2*pi*(F/Fs)*n);
hold on;
stem(n.*ts,f_digital);
```

- **Ponto 3: Réplicas espectrais**

A ideia deste ponto é mostrar a réplica espectral. Para isto, considere um sistema cuja taxa de aquisição é Fs=6.000 amostras por segundo. Se considerarmos a frequência de 1kHz, as frequências de 7k (1k+Fs), 13k(1k+2Fs), 19k(1k+3Fs) e assim por diante, passam pelos mesmos pontos que o sinal de 1kHz se confundindo com eles e gerando o indesejado fenômeno de aliasing.

```
%% inicializacoes
F1 = 1000;          %freq analogica
Fs = 6000;         %freq aquisicao
ts = 1/Fs;        %tempo aquisicao
num_ciclos = 1;  %(equivale a 3 ciclos)
t = 0:1e-6:(num_ciclos*(1/F1)); %tempo simulacao
n = 0:round(num_ciclos*(Fs/F1));        %numero amostras

%% parte analogica
f_analogico1 = sin(2*pi*F1*t);
f_analogico2 = sin(2*pi*7000*t);
f_analogico3 = sin(2*pi*13000*t);

plot(t,f_analogico1,'r');
hold on;
plot(t,f_analogico2,'b');
plot(t,f_analogico3,'k');
legend('1k','7k','13k');

%% parte digital
f_digital = sin(2*pi*(F1/Fs)*n);
stem(n.*ts,f_digital);
```

- **Ponto 4: operação com sinais**

Soma: a soma de dois sinais em função de n pode ser feita usando a função sigadd(sinal1, n_do_sinal1, sinal2, n_do_sinal2) conforme exemplo abaixo:

```
x1 = [1 2 3 2 1];
n1 = [-2 -1 0 1 2];
x2 = [2 3 4];
n2 = [0 1 2];
[x3, n3] = sigadd(x1,n1,x2,n2);
stem(n3,x3);
```

Multiplicação: a multiplicação de dois sinais em função de n pode ser feita usando a função sigmult(sinal1, n_do_sinal1, sinal2, n_do_sinal2) conforme exemplo abaixo:

```
x1 = [1 2 3 2 1];
n1 = [-2 -1 0 1 2];
x2 = [2 -3 4];
n2 = [-1 0 1];
[x3, n3] = sigmult(x1,n1,x2,n2);
```

```
6    stem(n3,x3);
```

Deslocamento: o deslocamento (avanço ou atraso) de um sinal pode ser gerado através da função sigshift conforme mostra o exemplo:

```
1    x1 = [1 2 3 2 1];
2    n1 = [0 1 2 3 4];
3    [x2, n2] = sigshift(x1,n1,2); %mesmo que x1(n-2)
4    subplot(2,1,1); stem(n1,x1); title('Original');
5    subplot(2,1,2); stem(n2,x2); title('Atrasada 2 unid');
```

Reflexão: a reflexão em torno do eixo vertical é dada por y(n)=x(-n) e é implementada conforme mostra o exemplo:

```
1    x1 = [1 2 3 4 5];
2    n1 = [0 1 2 3 4];
3    [x2, n2] = sigfold(x1,n1);
4    stem(n2, x2);
```

Convolução: embora o Matlab tenha uma função própria para convolução (a conv), ela leva em conta que os sinais são sempre casuais (ou seja, x[n]=0 para n<0). Para evitar esta limitação, é aconselhável usar a função conv_m conforme ilustra exemplo abaixo.

```
1    x1 = [3, 11, 7, 0, -1, 4, 2];
2    n1 = [-3:3];
3    h = [2, 3, 0, -5, 2, 1];
4    nh = [-1:4];
5    [y, ny] = conv_m(x1,n1,h,nh);
6    stem(ny, y);
```

Parte 2:

DFT

Nesta prática pretende-se demonstrar o uso da transformada discreta de Fourier (DFT) e a principal função do Matlab para cálculo da transformada discreta de Fourier: a FFT.

- **Ponto 1: calcular manualmente a transformada**

Para calcular a DFT de um sinal manualmente utilize o código abaixo. Vale destacar que neste caso consideramos o sinal como causal (em n=0). Note que neste exemplo não se nota a preocupação em otimizar o código e se pode usar tanto a forma de Euler (linha 10) ou pela sua forma expandida (linha 11). Ambos casos conduzem a um mesmo resultado.

Código 3.1 - Cálculo manual da DFT

```
1   clear; clc;
2   sinal = [1 2 3 4 5]; %sinal
3   Fs = 1000; %taxa de aquisição
4   N = size(sinal,2); %num de amostras
5   y = zeros(1,N);
6   %% calcular a DFT
7   for m=0:(N-1)
8      acumulador = 0;
9      for n=0:(N-1)
10        acumulador = acumulador + sinal(n+1)*(cos(2*pi*n*m/N)-j*sin(2*pi*n*m/N));
11        %acumulador = acumulador + sinal(n+1)*exp(-j*2*pi*n*m/N);
12     end
13     y(m+1)=acumulador;
14  end
15  %% calcular eixo frequencias f:
16  m = 0:(N-1);
17  f = m*Fs/N;
18  %% plotar sinais
19  magX = abs(y);
20  angX = angle(y);
21  realX = real(y);
22  imagX = imag(y);
23  subplot(2,2,1); stem(f,magX); title('magnitude');
24  subplot(2,2,3); stem(f,angX); title('fase');
25  subplot(2,2,2); stem(f,realX); title('real');
26  subplot(2,2,4); stem(f,imagX); title('imaginario');
```

- **Ponto 2: fft**

O código ilustrado anteriormente é capaz de calcular a DFT mas não tem eficiência computacional pois não faz uso das propriedades de simetria de cálculo da DFT. Uma função computacionalmente mais eficiente para cálculo da DFT é a fft (*fast Fourier transform*). O código 3.2 ilustra seu uso.

Código 3.2 - Cálculo manual da DFT usando a função FFT do Matlab

```
1   clc; clear;
2   %% gera um sinal "sintetico" com 3 componentes de frequencia
3   Fs = 1000; %taxa de aquisicao do sinal
4   ts = 1/Fs;
5   N = 1000;  %numero amostras de amostras analisadas
6   t = (0:N-1)*ts;
7   sinal = 0.6*sin(2*pi*50*t) + 2*sin(2*pi*120*t) + 0.3*sin(2*pi*400*t);
8   ruido = 0.4*randn(size(t));
9   %% calcula a DFT usando a funcao fft
10  y = fft(sinal+ruido);
11  y = y/N; %se desejar pode-se dividir por N para "acomodar" os calculos
12  %% calcular o eixo das frequencias
13  m = 0:(N-1);
14  f = m*Fs/N;
15  y = y(1:N/2); %pegando so a primeira metade já que a outra é cópia
```

```
16  f = f(1:N/2); %pegando so a primeira metade já que a outra é cópia
17  %% calcular a potência do espectro em db
18  magnitude = abs(y);
19  fase = angle(y);
20  f_ref = max(magnitude); %escolhe o maior valor para ser a referencia para normalizacao
21  y_db = 20*log10(magnitude/f_ref);  %lembre que, por exemplo 0db = 1 unidade
22  %% plotar
23  subplot (3,1,1); plot(f ,magnitude); title('magnitude espectro'); xlabel('freq(Hz)');
24  ylabel('Amplitude');
25  subplot (3,1,2); plot(f ,y_db); title('potencia espectro');
26  subplot (3,1,3); plot(f ,fase); title('fase');
```

Note que ao sinal, na linha 10, foi inserido o ruído para cálculo da fft. Experimente executar o código usando a adição e sem a adição do ruído ao sinal para ver o efeito do ruído no espectro. Note ainda a amplitude do ruído que pode alcançar até 0.4 unidades (linha 8) enquanto que a terceira componentes espectral de 400Hz (ver linha 7) tem amplitude máxima de 0.3 unidades. Mesmo assim ela é mais perceptível que o ruído uma vez que este tem pouca correlação com funções de seno/cosseno. Observe ainda o quanto a função logarítmica amplifica as pequenas componentes e atenua as grandes.

- **Ponto 3: usando janelas para calcular a FFT de sinais**

Para diminuir os efeitos de diminuição do leakage do cálculo de uma DFT, usamos funções matemáticas (que chamamos de janelas) que são multiplicadas pelo sinal a ser analisado. O código abaixo mostra o uso de diferentes janelas. Ao lado do código pode ser visto o nome de diferentes janelas. Note, na linha 4, que algumas delas exigem parâmetros específicos.

Código 3.3 - Cálculo manual da DFT usando a função FFT do Matlab

```
1  N = 100;
2  janela1 = window(@blackmanharris,N);
3  janela2 = window(@hamming,N);
4  janela3 = window(@gausswin,N,2.5);
5  wvtool(janela1, janela2, janela3);
```

```
@barthannwin
@bartlett
@blackman
@blackmanharris
@bohmanwin
@chebwin
@flattopwin
@gausswin
@hamming
@hann
@kaiser
@nuttallwin
@parzenwin
@rectwin
@taylorwin
@triang
@tukeywin
```

Para ilustrar o uso de uma janela de Hanning, podemos proceder como no código 3.4 (que é uma versão adaptada do código 3.2). Veja a diferença da DFT de um sinal janelado e um "puro" (não janelado).

Código 3.4 - Uso de janelas para cálculo da DFT visando diminuir o leakage

```
1  clc; clear;
2  %% gera um sinal "sintetico"
3  Fs = 200;
4  ts = 1/Fs;
5  N = 300;
6  t = (0:N-1)*ts;
7  sinal = 0.6*sin(2*pi*13*t);
8  %% cria um segundo sinal "janelado"
9  janela = window(@hann,N);
10 sinal_jan = sinal'.*janela;
```

```
%% calcula a DFT usando a funcao fft
y = fft(sinal);
y_jan = fft(sinal_jan);
%% calcular o eixo das frequencias
m = 0:(N-1);
f = m*Fs/N;
y = y(1:N/2);
y_jan = y_jan(1:N/2);
f = f(1:N/2);
%% calcular a potência do espectro em db
magnitude = abs(y);
magnitude_jan = abs(y_jan);
f_ref = max(magnitude);
f_ref_jan = max(magnitude_jan);
y_db = 20*log10(magnitude/f_ref);
y_db_jan = 20*log10(magnitude_jan/f_ref_jan);
%% plota
subplot (3,2,1); plot(t ,sinal); title('sinal nao janelado');
subplot (3,2,3); stem(f ,magnitude); title('magnitude espec sem jan'); ylim([0 max(magnitude)]);
subplot (3,2,5); plot(f ,y_db); title('potencia espectro sem janela');
subplot (3,2,2); plot(t ,sinal_jan); title('sinal janelado');
subplot (3,2,4); stem(f ,magnitude_jan); title('magnitude espec com jan'); ylim([0 max(magnitude)]);
subplot (3,2,6); plot(f ,y_db_jan); title('potencia espectro com janela');
```

- **Ponto 4: diferença entre resolução espectral e densidade espectral**

Para ilustrar a diferença entre resolução espectral e densidade espectral utilizemos um exemplo. Considere o sinal $x[n]=\cos(0.48\pi n)+\cos(0.52\pi n)$. Calcula a DFT do sinal considerando

A) um número de amostras N = 10 e preencha as outras 90 amostras com zero chegando a N=100.

B) um número de amostras N = 100 sem preenchimento com zero.

Código 3.5 - Código para solução dos dois itens anteriores. Observe que é usada a função DFT da biblioteca empregada neste apostila.

```
clc; clear;
n = [0:99];
Fs = 100;
x = cos(0.48*pi*n) + cos(0.52*pi*n);
%% calcula DFT com N=10 e plota
n1 = [0:9];
sinal1 = x(1:10);
Y1 = dft(sinal1, 10);
magY1 = abs(Y1(1:6));
N1 = 5;
m1= 0:5;
f1= m1*Fs/N1;
subplot(2,3,1); stem(sinal1); xlabel('n'); title('N=10')
subplot(2,3,4); stem(f1, magY1); xlabel('freq(Hz)');
%% calc DFT com N=10+90 zeros e plota (melhor resol espectro)
sinal2 = [x(1:10) zeros(1,90)]; %preencheu sinal que essencialmente eh o mesmo
Y2 = dft(sinal2,100);
magY2 = abs(Y2(1:50));
N2 = 50;
m2= 1:50;
f2= m2*Fs/N2;
subplot(2,3,2); stem(sinal2); xlabel('n'); title('N=10+90 zeros com boa resol. espec.')
subplot(2,3,5); stem(f2, magY2); xlabel('freq(Hz)');
%% calc DFT com N=100
sinal3 = x(1:100);
Y3 = dft(sinal3,100);
magY3 = abs(Y3(1:50));
N3 = 50;
m3= 1:50;
f3= m3*Fs/N3;
subplot(2,3,3); stem(sinal3); xlabel('n'); title('N=100 com boa densid. espec.')
subplot(2,3,6); stem(f3, magY3); xlabel('freq(Hz)');
```

Parte 3:

Projeto de filtros FIR

Nesta prática pretende-se demonstrar (i) o projeto de filtros de resposta finita (FIR - finite impulse response) através do cálculo de seus coeficientes e (ii) a implementação do filtro. Pra isto usamos a técnica de "janelas".

Resumo pontos abordados nesta parte	
Ponto 1	*Execução de uma filtragem usando função "conv"*
Ponto 2	*Calculando coeficientes FIR fase a fase*
Ponto 3	*Calculando coeficientes FIR usando funções fir1 e fir2*
Ponto 4	*Calculando coeficientes FIR usando Parks-McClellan*

➢ Ponto 1: Implementação de filtros FIR

Para implementar um filtro FIR é necessário que se tenha em mãos os coeficientes do filtro. Por isto, antes é necessário aprender a projetar um filtro que é o que faremos nos próximos tópicos. Por enquanto, vamos imaginar que já temos os coeficientes prontos e nossa intenção se resume a implementar o filtro considerando que o projeto (que é a designação dos coeficientes do filtro) já esteja pronta. O Código 4.1 mostra como é feita a implementação de um filtro FIR passa baixas com coeficientes já dados (liha 2 e 3). Note que a execução do filtro é feita por uma convolução da entrada com a resposta impulsiva h(n) do filtro na linha 17.

Código 4.1 - Implementação de um filtro FIR dados seus coeficientes.

```
1   clc; clear;
2   h = [0.4375, 0.3142, 0.0625, -0.0935, -0.0625, 0.0418, 0.0625, -0.0124, -0.0625, -0.0124,
3   0.0625, 0.0418, -0.0625, -0.0935, 0.0625, 0.3142];
4   n_h = 0:15;
5   %% PARTE 1: Gera um sinal analógico (entrada_analogica) e sua versão discretizada
6   (entrada_discretizada)
7   Fs = 20000;              % taxa de amostragem: 20mil amostras/seg
8   t = 0:(1/Fs):0.02;       % amostragem de 0,02seg
9   N = max(t)/(1/Fs);
10  n = 0:N;    % quantidade de amostras da entrada para filtrar
11  %entrada_analogica = sin(2*pi* 200*t) + sin(2*pi*25*t) + sin(2*pi* 5000*t) + sin(2*pi*
12  9000*t);
13  entrada_discretizada = sin(2*pi*200.*n/Fs) + sin(2*pi*25.*n/Fs) + sin(2*pi*5000.*n/Fs) +
14  sin(2*pi*9000.*n/Fs);
15
16  %% PARTE 2: convolui entrada com h
17  saida = conv(entrada_discretizada, h);
18
19  %% PARTE 3: PLOTA RESULTADOS
20  % calcula espectro entrada
21  fft_sinal_entrada = fft(entrada_discretizada)/N;
22  f_entrada = n.*(Fs/N);
23  % calcula espectro saida
24  N3 = size(saida,2);
25  fft_sinal_saida =   fft(saida)/N3;
26  n3 = 0:size(fft_sinal_saida,2)-1;
27  f_saida = n3.*(Fs/N3);
28  % calcula espectro filtro
29  fft_resp_filtro = fft(h);
30  n4 = 0:size(fft_resp_filtro,2)-1;
31  N4 = size(n4,2);
32  f_h = n4.*(Fs/N4);
33  %plota espectros
```

```
34  subplot(3,1,1); stem(n_h,h); xlabel('n'); title('Coef. filtro');
35  subplot(3,1,2); plot(f_entrada(1:N/2), abs(fft_sinal_entrada(1:N/2)));
36                  hold on;
37                  plot(f_saida(1:N3/2), abs(fft_sinal_saida(1:N3/2)),'r');
38                  plot(f_h(1:N4/2), abs(fft_resp_filtro(1:N4/2)),'g');
39                  legend('entrada','saída','resp. impulsiva');
40                  xlabel('Freq (Hz)');
41                  title('Espectros dos sinais e filtro')
42  %plota sinais
43  subplot(3,1,3); plot(n, entrada_discretizada);
44                  hold on;
45                  plot(n3, saida,'r');
46                  legend('entrada','saída');
47                  xlabel('n');
48                  title('Sinais discretos do filtro');
```

Uma outra forma muito usual de se executar uma filtragem no Matlab é usando a função "filter". Para isto, basta substituir a linha código "`saida = conv(entrada_discretizada, h);`" do exemplo anterior por: "`saida = filter(h,1,entrada_discretizada);`"

- **Ponto 2: Projetar um filtro FIR pelo método de janelas**

Projetar um filtro FIR usando o método de janelas é uma tarefa relativamente fácil. No exemplo do código 4.2 dividimos o código em oito partes para facilitar seu entendimento. Na parte 1 (linhas 3 a 7) especificamos as características do filtro. São elas:

- Fs: indica a taxa de aquisição máxima do sinal de entrada do filtro;
- Fc: indica a frequência de corte (em Hertz);
- N: número de coeficientes do filtro que pretendemos projetar;
- Num_Coef_filtro: dos N coeficientes calculados, devemos usar apenas parte deles que será designado por esta variável Num_Coef_filtro.

Na parte 2 (linhas 10 a 16) especificamos o espectro ideal para o filtro baseado nos valores coletados na parte 1. A partir deste espectro idealizado encontramos na parte 3 (linha 19) quais devem ser os coeficientes ideais do filtro para implementação deste espectro idealizado. Note que se o filtro é passa alta (linha 20) devemos multiplicar os coeficientes encontrados por $\cos(2\pi n(Fs/2)/Fs)=\cos(\pi n)=[1,-1,1,-1,...]$. Isto transforma o filtro passa-baixas em passa-altas. Já na linha 25 o código testa se foi escolhido um filtro passa-faixas para transformar o filtro passa-baixas em um passa-faixas através do deslocamento de frequência feito pela multiplicação dos coeficientes do filtro passa-baixa por $\cos(2\pi n f_{cc}/F_s)$ onde fcc indica a frequência de centro da banda de passagem.

Na parte 4 cuidados para gerar um gráfico de coeficientes simétrico em relação ao eixo vertical pois a ifft só estima metade da função sync. A outra metade deve ser preenchida através de espelhamento. Na parte 5 temos a opção de usar uma janela aos coeficientes para atenuar o ripple da banda passante. Aplicar uma janela é importante quando Num_Coef_filtro < N pois nesta situação é gerado um ripple que pode ser atenuado por algumas funções de janelamento.

Finalmente, nas partes 6 a 8 geramos um sinal sintético com 4 componentes de frequência para testar o filtro e calculamos os espectros do sinal sintético de entrada, o espectro da resposta impulsiva do filtro e o espectro do sinal de saída. Faça testes para ver os efeitos de filtragem.

Código 4.2 - Projeto de um filtro FIR passa-baixas, passa-latas e faixas.

```
1   clc; clear;
```

```
%% PARTE 1: dados do filtro
Fs = 20000;            %taxa de amostragem
fc = 2000;             %frequencia de corte do filtro
N =  64;               %Quantidade total de coeficientes filtro
Num_Coef_Filtro = 64;  %Qtos coeficientes vou usar do total
tipo_filtro = input('\nTipo filtro(1=passa-baixas; 2=passa-altas; 3=passa-faixa) =');

%% PARTE 2: gera o espectro ideal do filtro
H = zeros(1,N);             %resposta em freq ideal do filtro
f_resol = Fs/N;             %calcula resolução frequencia
m_corte = round(fc/f_resol)+1;   %estima indice "m" de fc
H = [ones(1, m_corte + 1) zeros(1, (N/2-m_corte))  zeros(1, (N/2-m_corte))  ones(1, m_corte
- 1)]; %gera espectro ideal do filtro
m = 0:N-1;
f = m.*f_resol;             %define o eixo de frequencias do grafico 1

%% PARTE 3: gera todos coef. ideais do filtro
h_ideal = ifft(H);
if (tipo_filtro==2) %se passa-alta, multiplicar coef por [1,-1,1,-1,...]
    n = 0:N-1;
    deslocamento_f = cos(2*pi*n.*(Fs/2)/Fs);
    h_ideal = h_ideal.*deslocamento_f;
end
if (tipo_filtro==3) %se passa-faixa, multiplicar coef por modulacao
    n = 0:N-1;
    fcc = input('\nDigite em Hz frequência central =');
    deslocamento_f = cos(2*pi*fcc.*n/Fs);
    h_ideal = h_ideal.*deslocamento_f;
end

%% PARTE 4: GERA SIMETRIA PAR NA FUNCAO SYNC DOS COEFICIENTES
h_ideal(N/2:N) = h_ideal(1:N/2+1);
for i=2:(N/2)
    h_ideal(i-1)= h_ideal(N-i+1);
end
inicio = N/2 - Num_Coef_Filtro/2 + 1;
fim    = N/2 + Num_Coef_Filtro/2;
h = real(h_ideal(inicio:fim)); %pega so parte dos coeficientes ideias do filtro

%% PARTE 5 (opcional): aplica janela
resposta = input('\nDeseja aplicar janela aos coef (1=sim; 2=nao)? = ');
if (resposta == 1)
    jan = window(@blackman,(fim-inicio)+1);
    h = h.*jan';
end

%% PARTE 6: testa a implementação filtro com sinal sintetico
N_sinal_sintetico = 400;     % quantidade de amostras do sinal sintetico
n1 = 0:N_sinal_sintetico-1;
entrada_discretizada = sin(2*pi*500.*n1/Fs) + sin(2*pi*2500.*n1/Fs) + sin(2*pi*5000.*n1/Fs)
+  sin(2*pi*8000.*n1/Fs);
saida = conv(entrada_discretizada, h);
N_sinal_saida = size(saida,2);
N_resp_impulsitva = Num_Coef_Filtro;

%% PARTE 7: calcula espectros sinal entrada, saida e h(n)
fft_sinal_entrada = fft(entrada_discretizada)/N_sinal_sintetico;
fft_sinal_saida   = fft(saida)/N_sinal_saida;
fft_resp_filtro   = fft(h);
f_entrada = n1.*(Fs/N_sinal_sintetico);
n3 = 0:size(fft_sinal_saida,2)-1;
f_saida   = n3.*(Fs/N_sinal_saida);
n2 = 0:size(fft_resp_filtro,2)-1;
f_h = n2.*(Fs/N_resp_impulsitva);

%% PARTE 8: plota
subplot(2,2,1); stem(f,H); title('H(f) idealizado'); xlabel('f(Hz)')
subplot(2,2,2); stem(real(h_ideal)); xlabel('n'); hold on; stem([inicio:fim],h,'-.r');
title('Coeficientes h(n) do filtro'); ylabel('Amplitude'); xlabel('n');
legend('Todos','Selecionados');
subplot(2,2,3); plot(f_entrada(1:N_sinal_sintetico/2),
abs(fft_sinal_entrada(1:N_sinal_sintetico/2)));
                hold on;
                plot(f_saida(1:N_sinal_saida/2),
abs(fft_sinal_saida(1:N_sinal_saida/2)),'r');
                plot(f_h(1:N_resp_impulsitva/2),
abs(fft_resp_filtro(1:N_resp_impulsitva/2)),'g');
```

```
                legend('entrada','saída','resp. impulsiva');
                xlabel('Freq (Hz)');
                title('Espectros dos sinais e filtro')
subplot(2,2,4); plot(n1, entrada_discretizada);
                hold on;
                plot(n3, saida,'r');
                legend('entrada','saída');
                xlabel('n');
                title('Sinais discretos do filtro');
```

Uma forma muito interessante de se avaliar a resposta impulsiva de um filtro é usando a função "freqz" do Matlab. O argumento desta função são os coeficientes h(n) do filtro e o resultado é a magnitude da resposta em frequência do filtro (designado como "mag" no código 4.3) e as correspondentes frequências normalizadas por π. Assim, a maior frequência tem valor 1 e indica a frequência de Nyquest que em nosso exemplo é Fs/2=20.000/2=10.000Hz

Código 4.3 - Visualização da resposta em frequência de um filtro

```
[mag,f] = freqz(h);
plot(f/pi,abs(mag))
```

- **Ponto 3: Projetar um filtro FIR usando a função do Matlab FIR1 e FIR2.**

A função FIR1 do Matlab tem um dos seguintes formatos:

$$\text{coeficientes} = \text{fir1(n,Wn)}$$
$$\text{coeficientes} = \text{fir1(n,Wn,'ftype')}$$
$$\text{coeficientes} = \text{fir1(n,Wn,window)}$$
$$\text{coeficientes} = \text{fir1(n,Wn,'ftype',window)}$$

Onde Wn indica a faixa de frequências que determinam o filtro, n a ordem, ftype o tipo (que pode ser 'high' ou 'stop') e window os coeficientes da janela que pode ser opcional. Observe os exemplos do código 4.4. A função FIR1 usa o método de projeto conhecido como "projeto baseado por resposta ao impulso".

Código 4.4 – Cálculos coeficientes filtro FIR usando função FIR1

```
ordem = 50;
resol_plot_freq = 512;

%Exemplo 1a: passa-baixas (frequência normalizada)
coef = fir1(ordem, 0.3);
freqz(coef,1,resol_plot_freq)

%Exemplo 1b: passa-baixas (frequência nominal considerando Fs=20k)
fc = 3000;
Fs = 20e3;
coef = fir1(ordem, fc/(Fs/2));
[H , freq] = freqz(coef,1,resol_plot_freq, 20e3);
plot(freq,abs(H));



%Exemplo 2: passa-altas
coef = fir1(ordem, 0.4,'high');
freqz(coef,1,resol_plot_freq)

%Exemplo 3: passa-altas com janela de Hanning
coef = fir1(ordem, 0.6, 'high', hann(ordem+1));
freqz(coef,1,resol_plot_freq)
```

```
%Exemplo 4: passabanda
coef = fir1(ordem, [0.3 0.5]);
freqz(coef,1,resol_plot_freq)

%Exemplo 5: rejeita-banda
coef = fir1(ordem, [0.3 0.7], 'stop');
freqz(coef,1,resol_plot_freq)
```

No código 4.4, deve-se escolher frequências normalizaas. Isto quer dizer que a maior frequência que pode ser representada pelo sistema, que equivale a Fs/2 (segundo critério de Nyquest) é denominada de 1. Assim, se desejamos projetar um filtro para cortar em 3000Hz usando um sistema com Fs=20.000, sabe-se que a máxima frequência de 10.000Hz equivale a 1. Logo, a frequência de 3.000Hz equivale 3.000/10.000 = 0,3. A função freqz analisa a resposta em frequência dos coeficientes.

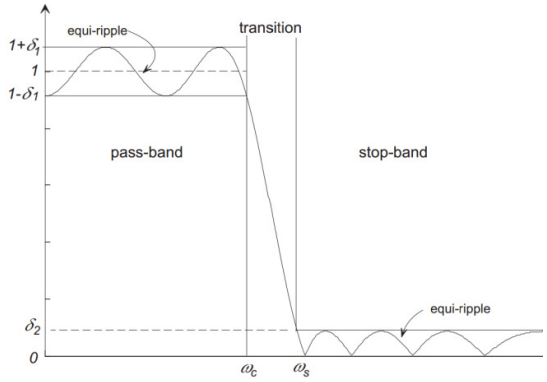
Um outro tipo de projeto dos coeficientes do filtro FIR é conhecido como "amostragem baseada em frequência" e é implementada pela função FIR2 conforme ilustra o código 4.5. A função FIR2 tem geralmente o formato fir2(n,f,m,window) onde n indica o ordem do filtro, f as frequências do filtro, m as respectivas amplitudes que devem ter as frequências especificadas pelo vetor f e 'window' o tipo de janela usado para os n+1 coeficientes do filtro (o uso de janelas é opcional e pode ser deixado em branco conforme ilustra alguns dos exemplos do código 4.5).

Código 4.5 – Cálculos coeficientes filtro FIR usando função FIR2

```
1   ordem = 50;
2   resol_plot_freq = 512;
3
4   %Exemplo 1: passa-baixas
5   f = [0, 0.3,  0.4,  1]; m = [1, 1, 0, 0];
6   coef = fir2(ordem, f, m);
7   freqz(coef,1,resol_plot_freq);
8
9   %Exemplo 2: passa-altas
10  f = [0, 0.5,  0.65,  1]; m = [0, 0, 1, 1];
11  coef = fir2(ordem, f, m);
12  freqz(coef,1,resol_plot_freq);
13
14  %Exemplo 3: passa-altas com janela
15  f = [0, 0.5,  0.65,  1]; m = [0, 0, 1, 1];
16  coef = fir2(ordem, f, m, hann(ordem+1));
17  freqz(coef,1,resol_plot_freq);
18
19  %Exemplo 4: passa-banda
20  f = [0, 0.4, 0.5, 0.7, 0.8, 1]; m = [0, 0, 1, 1, 0, 0];
21  coef = fir2(ordem, f, m);
22  freqz(coef,1,resol_plot_freq);
23
24  %Exemplo 5: rejeita-banda
25  f = [0, 0.4, 0.5, 0.7, 0.8, 1]; m = [1, 1, 0, 0, 1, 1];
26  coef = fir2(ordem, f, m);
27  freqz(coef,1,resol_plot_freq);
```

- **Ponto 4: Projetar um filtro FIR pelo método de Parks-McClellan**

  É uma das técnicas mais usadas atualmente para se encontrar coeficientes pois é uma técnica otimizada. Seu algoritmo é complexo por isto vamos nos ater somente a ensinar como proceder para usar as funções do Matlab para encontrar os valores dos coeficientes do filtro FIR. Para isto devemos lembrar os principais parâmetros de um filtro como ilustra a figura abaixo.



Baseados no que vemos na figura anterior, devemos especificar estes principais parâmetros para se determinar os coeficientes. Para isto podemos usar a função firpm do Matlab como ilustra o Código 4.4. Para projetar um filtro passa-faixas devemos usar as especificações das linhas 2 e 3 e um filtro passa-baixas as linhas 6 e 7. Utilize a função freqz para ver os resultados e entender como funciona a especificação destes filtros. Lembre-se que a frequência deve ser normalizada. Ou seja, a frequência 1 indica a frequência de Nyquest.

Código 4.6 - Ilustração do uso da função firpm que executa o algoritmo de Parks-McClellan para encontrar coeficientes de filtro FIR.

```
1   %coeficientes para filtro passa-faixa:
2   % f = [0 0.4 0.44 0.56 0.6 1];
3   % a = [0 0 1 1 0 0];
4
5   %coeficientes para filtro passa-baixas:
6   f = [0 0.4 0.5 1];
7   m = [1 1 0 0];
8
9   coef = firpm(32,f,m);
10  freqz(coef);
```

Note no exemplo anterior que escolhemos 32 coeficientes. Mas seria este valor correto? Geralmente, o projeto de um filtro envolve as especificações da atenuação, ripple e banda de transição. Assim, devemos inserir estes valores para descobrir de quantos coeficientes precisamos para conseguir estes valores. Assim, o modo mais eficiente de se usar a função firpm é antes determinar os parâmetros ótimos do filtro usando a função firpmord como exemplifica o código 4.5. Para isto considere (segundo figura anterior) wc=1500Hz; ws=2000Hz; δ_1=0.01 e δ_2=0.1 com Fs=8000 amostras por segundo. Note que usando estes argumentos usamos a função "firpmord" para encontrar os melhores valores para a ordem do filtro e os pontos chaves de frequência do filtro. Nas linhas 4 e 5 plotamos a resposta em frequência do filtro.

Código 4.7 - Uso otimizado do filtro FIR usando Parks-McClellan – Exemplo de um filtro passa-baixas

```
1   Fs = 8000;
2   [n,fo,ao,w] = firpmord([1500 2000],[1 0],[0.01 0.1],Fs);
3   coef = firpm(n,fo,ao,w);
    freqz(coef,1,512)
```

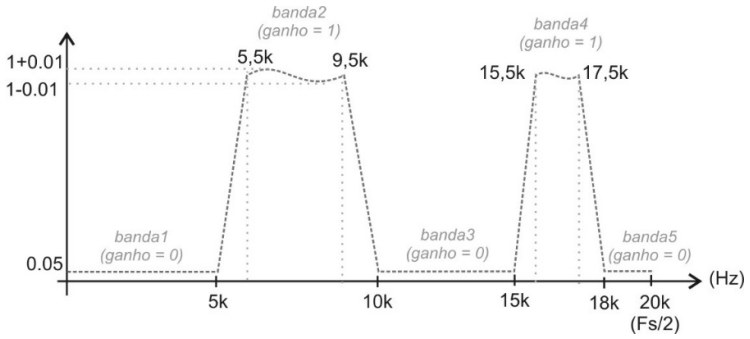
No código da sequência é ilustrado o emprego da mesma função para um filtro passa-banda. Observe que diferentemente das funções fir1 e fir2, a função firpmord emprega um modo diferente para representar os ganhos e atenuações nas bandas (de passagem ou rejeição). No caso do exemplo abaixo,

temos três bandas. A primeira delas é de rejeição (pois o ganho é próximo de 0 conforme especificado no código) e vai de 0 a 1500Hz. A segunda é de passagem (pois tenho ganho 1) e vai de 2000 a 3000 Hz. E a terceira banda é de rejeição e vai de 3500 Hz até $F_s/2 = 8000/2 = 4000$ Hz.

Código 4.8 – Exemplo de um filtro passa-banda Parks-McClellan.

```
Fs = 8000;
k1 = 0.1;    % atenuacao primeira banda de rejeicao
k2 = 0.05;   % ripple da banda de passagem
k3 = 0.1;    % atenuacao segunda banda de rejeicao
[n, fo, ao, w] = firpmord([1500 2000 3000 3500], [0 1 0], [k1, k2, k3], Fs);
b  = firpm(n, fo, ao, w);
%% calcula o eixo da freq
[espectro, w] = freqz(b,1,512); %retorna o espectro em 'espectro' e o eixo de frequen em w
freq = (w/pi); %normaliza as frequencias de 0 ate 1 onde 1=Fs/2 (w está de 0 até pi);
freq = freq*(Fs/2); %pego o vetor freq normalizado e multiplico pela máx freq que é Fs/2
%% plota
subplot(2,1,1); plot(freq, 20*log10(abs(espectro))); title('escala dB'); grid on;
subplot(2,1,2); plot(freq, abs(espectro));           title('escala linear');  grid on;
```

No exemplo anterior, pode-se perceber que no vetor de frequências deve ser designado para cada transição as duas frequências que designam a banda de transição. Por exemplo, para projetar o filtro da figura abaixo, deve-se observar que ele tem 5 bandas e 4 bandas de transição. Como cada banda de transição é especificada por um par de frequências, teremos que especificar 8 frequências para a função firpmord. O código necessário para implementar este filtro é mostrado logo abaixo.



Código 4.9 – Exemplo do uso da função firpmord para o filtro ilustrado na figura.

```
Fs = 40000;
k1 = 0.05;    % atenuacao nas bandas de rejeicao
k2 = 0.01;    % ripple nas bandas de passagem
[n, fo, ao, w] = firpmord([5000 5500 9500 10000 15000 15500 17500 18000], [0 1 0 1 0], [k1
k2 k1 k2 k1], Fs);
b  = firpm(n, fo, ao, w);
%% calcula o eixo da freq
[espectro, w] = freqz(b,1,512);
freq = (w/pi);
freq = freq*(Fs/2);
%% plota
subplot(2,1,1); plot(freq, 20*log10(abs(espectro))); title('escala dB'); grid on;
subplot(2,1,2); plot(freq, abs(espectro));           title('escala linear');  grid on;
```

Parte 4:

Transformada Z em Matlab

O objetivo desta prática é aprender as funções do Matlab que podem ser utilizadas para analisar sinais e sistemas no espaço Z. Vale lembrar que geralmente um sinal ou um sistema pode ser descrito como uma relação de dois polinômios definidos na forma:

$$S(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \ldots + a_N z^{-N}} \tag{2.0}$$

Assim, os coeficientes b_M expressam o numerador desta razão e a_N os coeficientes do polinômio de denominador. Lembre-se desta convenção pois a inversão desta ordem pode originar um resultado errado. Os próximos "pontos" descrevem os principais recursos para se trabalhar com transformadas z.

- **Ponto 1: Determinação da transformada z**

  O Matlab não tem uma função "mágica" que se possa inserir uma equação de sistema ou sinal e automaticamente calcular sua transformada Z. Muitas vezes é preciso trabalhar matematicamente na expressão para fazer um arranjo e poder usar as funções do Matlab. O próximo exemplo ilustra isto.

  Exemplo: usando as propriedades da transformada Z e a tabelada Z, determine a transformada Z do sinal:

  $$x[n] = (n-2)(0{,}5)^{(n-2)} cos\left(\frac{\pi}{3}(n-2)\right) u(n-2)$$

  Solução: a análise do sinal revela que ele foi atrasado em 2 unidades. Aplicando a tabela de propriedades (ver propriedade de atraso do sinal) podemos reescrever:

  $$X(z) = z^{-2} . \left[n(0{,}5)^n cos\left(\frac{\pi}{3} n\right) u(n) \right]$$

  Uma nova análise na tabela de propriedades revela que um sinal multiplicado por "n", como é este o caso, deve ser derivado no espaço Z tal como:

  $$X(z) = z^{-2} . \left[-z \frac{d}{dz} \left\{ (0{,}5)^n cos\left(\frac{\pi}{3} n\right) u(n) \right\} \right] \tag{2.1}$$

  Aplicando a transformada Z (ver tabelada Z) nos termos contido entre chaves {.}, temos:

  $$Z\left\{ (0{,}5)^n cos\left(\frac{\pi}{3} n\right) u(n) \right\} = \frac{1 - \left(0{,}5 cos\frac{\pi}{3}\right) z^{-1}}{1 - 2\left(0{,}5 cos\frac{\pi}{3}\right) z^{-1} + 0{,}25 z^{-2}} = \frac{1 - 0.25 z^{-1}}{1 - 0.5 z^{-1} + 0{,}25 z^{-2}} \tag{2.2}$$

  Inserindo o resultado de (2.2) em (2.1), temos:

  $$X(z) = -z^{-1} . \frac{d}{dz} \left\{ \frac{1 - 0.25 z^{-1}}{1 - 0.5 z^{-1} + 0{,}25 z^{-2}} \right\}$$

  Aplicando a derivada e rearanjando os termos temos finalmente:

  $$X(z) = \frac{0.25 z^{-3} - 0.5 z^{-4} + 0.0625 z^{-5}}{1 - z^{-1} + 0{,}75 z^{-2} - 0.25 z^{-3} + 0.0625 z^{-4}}$$

Agora que calculamos a transformada do sinal e a colocamos no formato desejado ilustrado em (2.0), usaremos o Matlab da seguinte forma:

```
clear; clc;
%teste 1: usando na forma de eq. diferencas:
n = [0:7];
sinal = [(n-2).*(1/2).^(n-2).*cos(pi*(n-2)/3)].*[0 0 1 1 1 1 1 1]

%teste 2: usando transf z
b=[0, 0, 0, 0.25, -0.5, 0.0625]; %coef b do numerador
a=[1, -1, 0.75, -0.25, 0.0625]; %coef a do denominador
entrada = [1 0 0 0 0 0 0 0];
sinal = filter(b, a, entrada)
```

- **Ponto 2: trabalhando com a equação de diferenças**

Como visto na teoria de PDS, uma equação de diferenças pode representar o formato matemático de um determinado sinal de saída ou até mesmo expressar matematicamente como um sistema age em um sinal de entrada produzindo uma saída. Seu formato básico é dado por:

$$y(n) = \sum_{m=0}^{M} b_m x(n-m) - \sum_{k=1}^{N} a_k y(n-k)$$

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{m=0}^{M} b_m x(n-m)$$

Assim, a_k são os coeficientes que multiplicam y(.) e b_m os que multiplicam x(.). A função filter do Matlab pode ser usada para resolver numericamente equações diferença como mostra o exemplo abaixo.

Exemplo: dada a equação diferença abaixo, calcule:

$$y(n) = y(n-1) - 0.9y(n-2) + x(n)$$

a) a resposta impulsiva h(n) do sistema entre o intervalo n=-20 a 100;

b) a saída do sistema considerando uma entrada de degrau unitário entre o intervalo n=-20 a 100. O sistema é estável ?

Solução:

a) reescrevendo a equação para o modelo padrão, temos:

$$y(n) - y(n-1) + 0.9y(n-2) = x(n)$$

Que em Matlab é escrito como:

```
b=[1];
a=[1, -1, 0.9];
x = impseq(0, -20, 120);
n = [-20:120];
h = filter (b, a, x);
subplot(2,1,1); stem(n, h);  title('resposta impulsiva h(n)');
```

b) Para achar a saída, façamos:

```
x = stepseq(0, -20, 120);
y = filter (b, a, x);
```

```
3   subplot(2,1,2); stem(n, y);  title('resposta ao degrau');
```

Para verificar se o sistema é estável, usaremos o critério de que uma entrada limitada (ou somável), como é este o caso, deve também produzir uma saída limitada (somável). Assim, basta fazer `sum(abs(y))` que resultará em um valor próximo de 14.8 concluindo que o sistema é estável.

Outra forma de fazer esta análise é aplicando a transformada Z na equação do sistema (que será visto mais a frente) como:

$$Z\left(\frac{y(n)}{x(n)}\right) = H(z) = \frac{1}{1 - 1z^{-1} + 0.9z^{-2}}$$

Analisando as raízes de denominador da relação anterior (que chamaremos mais à frente de polos), vemos pelo código abaixo que os polos (que são valores complexos conjugados) tem valores positivos em sua parte real e menores que 1 o que indica que o sistema é estável quando se aplica esta determinada entrada à equação deste sistema.

```
1   z = roots(a)
2   z =
3      0.5000 + 0.8062i
       0.5000 - 0.8062i
```

- **Ponto 3: Decomposição (resíduos) de uma função**

Para estimar a função inversa de uma transformada Z, geralmente deseja-se fracionar a razão de polinômios que é representada na forma de (2.0). Para isto, considere o exemplo abaixo.

Exemplo: Considere a função abaixo. Faça sua decomposição em resíduos da forma mais simples possível.

$$X(z) = \frac{z}{3z^2 - 4z + 1}$$

Solução: para proceder com a solução do sistema, é necessário reescrever a função de tal forma que só tenhamos expoentes de z negativo. Assim, fazemos:

$$X(z) = \frac{z}{3z^2 - 4z + 1} \cdot \frac{z^{-2}}{z^{-2}} = \frac{z^{-1}}{3 - 4z^{-1} + z^{-2}}$$

Usando o Matlab temos:

```
1   b = [0, 1];
2   a = [3, -4, 1];
3   [R, p, C] = residuez(b,a)
4   R =
5       0.5000
6      -0.5000
7   p =
8       1.0000
9       0.3333
10  C =
11        []
```

Este resultado quer dizer que:

$$X(z) = \frac{z}{3z^2 - 4z + 1} = \frac{0.5}{1 - 1.0z^{-1}} + \frac{-0.5}{1 - 0.333z^{-1}}$$

A forma simplificada ou residual permite uma fácil comparação com a tabelada Z e assim a consequente transformação do sinal do espaço Z para sua forma original. No código anterior,

R indica os resíduos (coeficiente do numerador), p indica os pólos (raízes do polinômio do denominador) e C as constantes que devem ser somadas aos termos encontrados.

É também possível pegar uma expressão já na forma residual e convertê-la na forma racional como mostra o código abaixo:

```
R = [0.5, -0.5];
p = [1.0, 0.3333];
C = [];
[b, a] = residuez(R, p, C)
b =
    0         0.3334
a =
    1.0000   -1.3333    0.3333
```

Que equivale a forma original mostrada no início desta solução.

$$X(z) = \frac{0 + 0.3334z^{-1}}{1.0 - 1.333z^{-1} + 0.3334z^{-2}}$$

- **Ponto 4: Computação da transformada inversa usando frações parciais**

É também comum a necessidade de se calcular a transformada inversa através da análise de frações parciais conforme ilustra o exemplo abaixo.

Exemplo: dada a função abaixo, determine a transformada inversa da equação abaixo

$$X(z) = \frac{1}{(1-0.9z^{-1})^2(1+0.9z^{-1})} \qquad \text{ROC: } |z|>0{,}9$$

Solução: para iniciar a solução, vamos usar a função poly para calcular o polinômio do denominador. Em seguida, usando o que foi aprendido no ponto anterior, vamos usar a função residuez para decompor a expressão polinomial. Veja o código:

```
b = [1];
a = poly( [0.9,  0.9,  -0.9] )
a =
    1.0000   -0.9000   -0.8100    0.7290
[R, p, C] = residuez(b, a)
R =
   0.2500 + 0.0000i
   0.5000 - 0.0000i
   0.2500 + 0.0000i
p =
   0.9000 + 0.0000i
   0.9000 - 0.0000i
  -0.9000 + 0.0000i
C =
     []
```

Isto gera a função:

$$X(z) = \frac{0.25}{1 - 0.9z^{-1}} + \frac{0.5}{(1 - 0.9z^{-1})^2} + \frac{0.25}{1 + 0.9z^{-1}}$$

$$= \frac{0.25}{1 - 0.9z^{-1}} + \frac{0.5z}{0.9}\frac{0.9z^{-1}}{(1 - 0.9z^{-1})^2} + \frac{0.25}{1 + 0.9z^{-1}}$$

Que, pela análise da tabelada Z (ver anexo), podemos inferir que x[n] vale:

$$x[n] = 0.25(0.9)^n u(n) \; + \; \frac{5}{9}(n+1)(0.9)^{n+1}u(n+1) \; + \; 0.25(-0.9)^n u(n)$$

- **Ponto 5: Computação da resposta em frequência, fase e plano Z de uma função**

A variável z é complexa e por isto possui um módulo e uma fase. Como é sabido, uma função racional no formato da equação (2.0) possui polos (p) e zeros (z) em um padrão similar a:

$$X(z) = \frac{(z \pm z_0)(z \pm z_1)\dots(z \pm z_M)}{(z \pm p_0)(z \pm p_1)\dots(z \pm p_N)}$$

É portanto importante conhecer o comportamento da localização dos polos e zeros além de como a magnitude e fase de z podem afetar o comportamento de X(z). O exemplo abaixo ilustra isto.

Exemplo: Para o sistema y(n)=0.9y(n-1)+x(n), determine:

(a) encontre a transformada z de y(n) e desenhe seus polos e zeros;

(b) plote a magnitude e fase de para quando $z=e^{jw}$ (ou seja: r=1 lembrando que $z=re^{jw}$);

(c) determine a resposta impulsiva h(n) do sistema a partir de sua equação de diferenças.

Solução:

a) a transformada Z pode ser calculada como:

$$y(n) - 0.9y(n-1) = x(n) \Rightarrow H(z) = \frac{1}{1 - 0.9z^{-1}}$$

Para plotar polos (contidos no vetor a) e zeros (contidos no vetor b) temos:

```
1  b = [1];
2  a = [1, -0.9];
3  zplane(b, a);
```

b) Para responder a pergunta (b), usaremos a função freqz tal como:

```
1  b = [1];
2  a = [1, -0.9];
3  [H, w] = freqz(b, a, 100);
4  subplot(2,1,1);  plot(w/pi, abs(H));
5  grid on;  xlabel('Freq');  ylabel('Mag');  title('resposta em magnitude');
6
7  subplot(2,1,2); plot(w/pi, angle(H)/pi);
8  grid on;  xlabel('Freq');  ylabel('Fase'); title('resposta em fase');
```

No código anterior, a constate 100 (linha 3) indica o número de componentes de frequências analisadas.

c) Para calcular h(n), basta encontrar a transformada inversa de H(z) que conhecemos. Analisando a tabelada Z, podemos chegar ao resultado:

$$H(z) = \frac{1}{1 - 0.9z^{-1}} \xrightarrow{inversa} h(n) = (0{,}9)^n u(n)$$

- **Ponto 6: plotando a superfície tridimensional de uma função transferência no espaço Z**

Às vezes, por questões didáticas, é interessante a visualização da transformada Z no espaço tridimensional. Para fazer isto, utilize o código abaixo.

```
1  %% digite coeficientes b/a:
2  b = [6 -10 2];
3  a = [1 -3 2];
4  %% plota figura em 3D:
5  [x, y] = meshgrid(-3:0.1:3);
6  z = x+y*j;
7  res = (polyval(fliplr(b), z))./(polyval(fliplr(a),z));
8  subplot(1,2,1); surf(x,y, abs(res));
9  %% plota vista superior:
10 subplot(1,2,2); imagesc(-3:0.1:3,-3:0.1:3, abs(res));
11 rectangle('curvature', [1 1], 'position', [-1 -1 2 2], 'edgecolor', 'w');
12 axis equal
```

Anexos da prática

A) Tabela de algumas funções da transformada Z

	Signal, $x(n)$	z-Transform, $X(z)$	ROC
1	$\delta(n)$	1	All z
2	$u(n)$	$\frac{1}{1-z^{-1}}$	$\|z\| > 1$
3	$a^n u(n)$	$\frac{1}{1-az^{-1}}$	$\|z\| > \|a\|$
4	$na^n u(n)$	$\frac{az^{-1}}{(1-az^{-1})^2}$	$\|z\| > \|a\|$
5	$-a^n u(-n-1)$	$\frac{1}{1-az^{-1}}$	$\|z\| < \|a\|$
6	$-na^n u(-n-1)$	$\frac{az^{-1}}{(1-az^{-1})^2}$	$\|z\| < \|a\|$
7	$(\cos \omega_0 n)u(n)$	$\frac{1-z^{-1}\cos\omega_0}{1-2z^{-1}\cos\omega_0+z^{-2}}$	$\|z\| > 1$
8	$(\sin \omega_0 n)u(n)$	$\frac{z^{-1}\sin\omega_0}{1-2z^{-1}\cos\omega_0+z^{-2}}$	$\|z\| > 1$
9	$(a^n \cos \omega_0 n)u(n)$	$\frac{1-az^{-1}\cos\omega_0}{1-2az^{-1}\cos\omega_0+a^2z^{-2}}$	$\|z\| > \|a\|$
10	$(a^n \sin \omega_0 n)u(n)$	$\frac{az^{-1}\sin\omega_0}{1-2az^{-1}\cos\omega_0+a^2z^{-2}}$	$\|z\| > \|a\|$

B) Tabela de propriedades da transformada Z

Property	Time Domain	z-Domain	ROC
Notation	$x(n)$	$X(z)$	ROC: $r_2 < \|z\| < r_1$
	$x_1(n)$	$X_1(z)$	ROC_1
	$x_2(n)$	$X_2(z)$	ROC_2
Linearity	$a_1x_1(n) + a_2x_2(n)$	$a_1X_1(z) + a_2X_2(z)$	At least the intersection of ROC_1 and ROC_2
Time shifting	$x(n-k)$	$z^{-k}X(z)$	That of $X(z)$, except $z=0$ if $k>0$ and $z=\infty$ if $k<0$
Scaling in the z-domain	$a^n x(n)$	$X(a^{-1}z)$	$\|a\|r_2 < \|z\| < \|a\|r_1$
Time reversal	$x(-n)$	$X(z^{-1})$	$\frac{1}{r_1} < \|z\| < \frac{1}{r_2}$
Conjugation	$x^*(n)$	$X^*(z^*)$	ROC
Real part	$\text{Re}\{x(n)\}$	$\frac{1}{2}[X(z) + X^*(z^*)]$	Includes ROC
Imaginary part	$\text{Im}\{x(n)\}$	$\frac{1}{2}j[X(z) - X^*(z^*)]$	Includes ROC
Differentiation in the z-domain	$nx(n)$	$-z\frac{dX(z)}{dz}$	$r_2 < \|z\| < r_1$
Convolution	$x_1(n) * x_2(n)$	$X_1(z)X_2(z)$	At least, the intersection of ROC_1 and ROC_2
Correlation	$r_{x_1x_2}(l) = x_1(l) * x_2(-l)$	$R_{x_1x_2}(z) = X_1(z)X_2(z^{-1})$	At least, the intersection of ROC of $X_1(z)$ and $X_2(z^{-1})$
Initial value theorem	If $x(n)$ causal	$x(0) = \lim_{z\to\infty} X(z)$	
Multiplication	$x_1(n)x_2(n)$	$\frac{1}{2\pi j}\oint_C X_1(v)X_2\left(\frac{z}{v}\right)v^{-1}\,dv$	At least, $r_{1l}r_{2l} < \|z\| < r_{1u}r_{2u}$
Parseval's relation	$\sum_{n=-\infty}^{\infty} x_1(n)x_2^*(n) = \frac{1}{2\pi j}\oint_C X_1(v)X_2^*(1/v^*)v^{-1}dv$		

Parte 5:

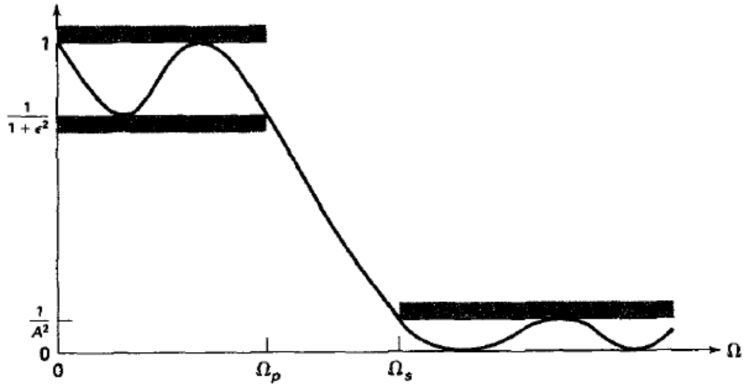
Projeto de filtros IIR

Nesta prática pretende-se demonstrar o projeto de filtros IIR a partir de transformações de filtros analógicos. Este projeto é realizado por etapas sequenciais. Deste modo, os pontos subsequentes representam as etapas necessárias para um projeto de um filtro IIR e sua implementação em cada um dos pontos abordados aqui. Destaca-se que este material é complementar as aulas teóricas e por isto não tem como objetivo ensinar todo o conteúdo de filtros IIR. Seu propósito é reforçar os conteúdos ensinados em sala e exemplificar como projetar um filtro em Matlab usando funções específicas.

➢ Ponto 1: Implementação de um filtro analógico passa-baixas Butterworth

Este ponto se propõem a descrever o projeto básico de um filtro Butterworth analógico passa-baixas usando o Matlab.

Um filtro analógico é especificado no domínio de Laplace (especificado pela variável complexa "s"). O primeiro passo é descobrir a ordem de um filtro que deve ser usado para satisfazer a determinados requerimentos. Estas especificações são novamente mostradas no gráfico abaixo.



Um filtro analógico é especificado no domínio de Laplace (representado pela variável complexa "s") e há várias maneiras de especificar um filtro. Geralmente elas são feitas por duas formas: (i) especificando a frequência de corte e ordem do filtro ou (ii) especificando a banda de transição, ganho na banda de passagem e atenuação na banda de rejeição. A primeira forma deve ser feita usando o código 5.1 onde se projeta um filtro com frequência corte 0.5 (lembrando que a máxima frequência é 1 que equivale a Nyquest) de ordem 3. Note a utilização da função butter do Matlab. O termo 'low' indica um filtro passa-baixas e o termo "s" indica que o filtro é feito no domínio do analógico de Laplace. Além de low podemos usar "high" e "stop" sendo este um rejeita-banda.

Código 5.1 - Projeto de um filtro Butterworth passa-baixas analógico a partir da frequência de corte e número de polos.

```
1  N = 3;
2  OmegaC = 2500/5000;  %freq de corte 2.500Hz com taxa de aquisição de 2x5.000Hz (normalizado)
```

```
3   [b,a] = butter(N,OmegaC,'low','s')
```

O resultado é: b=0.1250 e a=[1.0 1.0 0.50 0.1250] que matematicamente equivale a:

$$\frac{0.1250}{1.0s^3+1.0s^2+0.50s+0.1250}$$

A segunda forma de especificação deve ser determinada a banda de transição, o ganho na banda de passagem e atenuação na banda de rejeição. Este processo de projeto é mostrado no Código 5.2. Neste caso, usamos a biblioteca empregada nesta disciplina e a partir destas especificações determina-se a ordem do filtro e sua função matemática. Neste exemplo usamos uma banda de transição de 0.2π a 0.3π (lembrando que 1π é a frequência de Nyquest neste tipo de representação). O ripple da banda de passagem é de 7dB e a atenuação na banda de rejeição é 16dB neste exemplo.

Código 5.2 - Projeto de um filtro Butterworth passa-baixas analógico a partir da banda de transição, ripple e atenuação.

```
1   wp = 0.2*pi; % 0 a 0.2pi = banda passagem
2   ws = 0.3*pi; % 0.3pi a 1pi = banda rejeição
3   rp = 7;      %ripple banda passagem em dB
4   as = 16;     %atenuação mínima na banda rejeição em dB
5   ripple = 10^(-rp/20);    %descobre valor do ripple sem escala log
6   atenuacao = 10^(-as/20); %descobre valor da atenuação sem escala log
7   [b, a] = afd_butt(wp, ws, rp, as)
8   w = 0:pi/1000:pi;  %determina um eixo para freq
9   h = freqs(b,a,w);  %calcula a resposta em freq do filtro
10  plot(w/pi,abs(h));
```

O resultado é: b = 0.1238 e a = [1.0 0.9969 0.4969 0.1238] que matematicamente equivale a:

$$\frac{0.1238}{1.0s^3+0.9969s^2+0.4969s+0.1238}$$

➢ Ponto 2: Transformações de frequência de analógico (s) para digital (z)

Há várias formas de converter um filtro analógico especificado matematicamente no domínio "s" para um filtro no formato digital que deve ser especificado matematicamente no domínio "z". Uma destas técnicas de conversão é a (i) transformada de **invariância ao impulso** (que será ilustrada no código 5.3) e outra é a (ii) transformada bilinear. A transformação por invariância ao impulso utiliza a função "impinvar" do Matlab é ilustrada no código 5.3. Consideramos neste exemplo T=1 lembrando que **w=ΩT**.

Código 5.3 - Exemplo de conversão de uma equação no domínio "s" para o "z" usando o método de invariância ao impulso

```
1   %% PARTE 1: especificações projeto
2   wp = 0.2*pi; % 0 a 0.2pi = banda passagem
3   ws = 0.3*pi; % 0.3pi a 1pi = banda rejeição
4   rp = 7;      %ripple banda passagem em dB
5   as = 16;     %atenuação mínima na banda rejeição em dB
6   T = 1;       % taxa de amostragem da frequencia
7
8   %% PARTE 2: desenha filtro analógico
9   [b, a] = afd_butt(wp, ws, rp, as);
10  w = 0:pi/1000:pi;  %determina um eixo para freq
11
12  h = freqs(b,a,w);  %calcula a resposta em freq do filtro
13  subplot(2,1,1); plot(w/pi,abs(h)); title('resp freq analogico');
14
15  %% PARTE 3: digitaliza filtro usando inv ao impulso
16  [bz, az] = impinvar(b,a,T);
17  hz = freqz(bz,az,w);
18  subplot(2,1,2); plot(w/pi,abs(hz)); title('resp freq digital');
```

O resultado é: bz = [-0.0 0.0438 0.0314] e az = [1.0 -2.0233 1.4675 -0.3690] que matematicamente equivale a transformação:

$$\frac{0.1238}{1.0s^3+0.9969s^2+0.4969s+0.1238} \Rightarrow \frac{0.0438z^{-1}+0.0314z^{-2}}{1.0-2.0233z^{-1}+1.4675z^{-2}-0.3690z^{-3}}$$

Já a transformação bilinear emprega a função do matlab "bilinear" e para ver seu uso use o código 5.3 substituindo na linha 16 o nome "impinvar" por "bilinear". Ambas as funções servem a um mesmo propósito. Contudo, a transformação bilinear é geralmente mais usada por geralmente apresentar menos distorções na transformação do plano s para o círculo z.

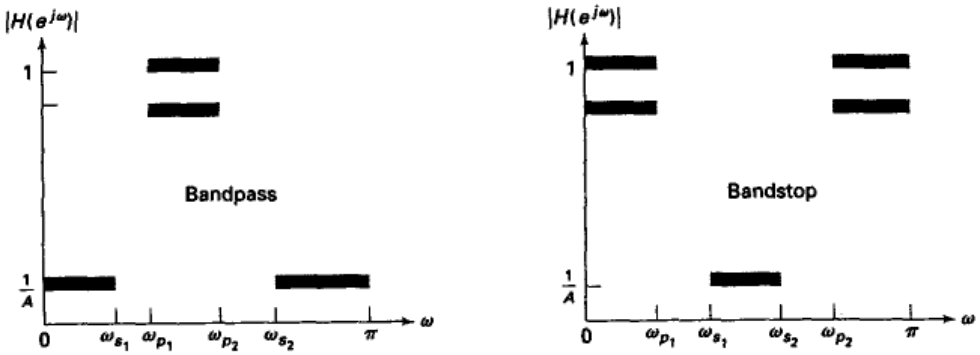
- **Ponto 3: Projeto de filtros passa-altas, rejeita-banda e passa-banda**

Para o projeto de filtros passa-altas, rejeita-bandas e passa-banda. Para projetar estes filtros usaremos a função butter do Matlab do seguinte modo:

- [b, a] = butter(N, wn, 'high') projeta um filtro passa-altas com frequência de corte wn (wn = wc = 3dB);
- [b, a] = butter(N, wn) projeta um filtro passa-bandas de ordem 2N se wn é um vetor tal que wn=[w1 w2] compreende a faixa de frequências que deve passar;
- [b, a] = butter(N, wn, 'stop') projeta um filtro rejeita-bandas de ordem 2N se wn é um vetor tal que wn=[w1 w2] e compreenda as faixas de frequências que devem ser rejeitadas;

Em todos os casos anteriores, **a frequência é dada em função de π** indicando que π refere-se a máxima frequência do sinal. Destaca-se também que as frequências especificadas anteriormente devem ter amplitudes próximas de -3dB (0,7 unidades).

Uma outra função importante é a **[N, wn]=buttord(wp, ws, rp, as)** onde especificamos a banda de transição especificadas pelas frequências [wp, ws], o ripple da banda de passagem (rp - em dB) e a atenuação nas banda de rejeição (as - em dB). O resultado da função é a determinação da ordem N do filtro e da frequência de corte wn. No caso desta função, para um filtro passa-bandas, wp e ws são vetores com dois elementos tal que: wp=[wp1, wp2] e ws=[ws1, ws2]. Para entender melhor estes parâmetros, veja as figuras abaixo. *É importante destacar que esta função considera frequências normalizadas na faixa [0 1] onde 1 indica a maior frequência.*



Código 5.4 - Código de um filtro passa-baixas de 0 a 3.400Hz com transição de 3400 a 4000Hz.

```
1  Fs = 40000;
2  wp= 3400/(Fs/2);
3  ws= 4000/(Fs/2);
4  [n, wn]=buttord(wp, ws, 1, 20);
5  [b, a]= butter(n,wn);
6  num_freq = 512;
```

```
7   freqz(b, a, num_freq, Fs);
```

Código 5.5 - Código de um filtro passa-altas de 4000Hz a 20000Hz sendo esta última a máxima frequência.

```
1   Fs = 40000;
2   wp= 4000/(Fs/2);
3   ws= 3400/(Fs/2);
4   [n, wn]=buttord(wp, ws, 1, 20);
5   [b, a]= butter(n,wn,'high');
6   num_freq = 512;
7   freqz(b, a, num_freq, Fs);
```

Código 5.6 - Código de um filtro passa-faixas de 4000 a 8000Hz com transições de largura 600Hz.

```
1   Fs = 40000;
2   ws1= 3400/(Fs/2);
3   wp1= 4000/(Fs/2);
4   wp2= 8000/(Fs/2);
5   ws2= 8600/(Fs/2);
6   [n, wn]=buttord([wp1 wp2], [ws1 ws2], 1, 20);
7   [b, a]= butter(n,wn);
8   num_freq = 512;
9   freqz(b, a, num_freq, Fs);
```

Código 5.7 - Código de um filtro rejeita-faixas que exclui as faixas de 4500 a 8000Hz e largura de transição de 500Hz.

```
1   Fs = 40000;
2   wp1= 4000/(Fs/2);
3   ws1= 4500/(Fs/2);
4   wp2= 8500/(Fs/2);
5   ws2= 8000/(Fs/2);
6   [n, wn]=buttord([wp1 wp2], [ws1 ws2], 1, 20);
7   [b, a]= butter(n,wn,'stop');
8   num_freq = 512;
9   freqz(b, a, num_freq, Fs);
```

➢ Ponto 4: Implementação do filtro

Para executar uma filtragem, basta ter em mãos os coeficientes designados por "b" e "a" e usar a função "filter" conforme ilustra o código 5.8 na linha 15.

Código 5.8 - Execução de um filtragem IIR.

```
1   Fs = 40000;
2   wp1= 4000/(Fs/2);
3   ws1= 3400/(Fs/2);
4   wp2= 6000/(Fs/2);
5   ws2= 6600/(Fs/2);
6   [n, wn]=buttord([wp1 wp2], [ws1 ws2], 1, 20);
7   [b, a]= butter(n,wn);
8   %% cria sinal sintético
9   t = 0:(1/Fs):0.02;
10  N = max(t)/(1/Fs);
11  n = 0:N;
12  entrada_discretizada = sin(2*pi*900.*n/Fs) + sin(2*pi*14500.*n/Fs) + sin(2*pi*5000.*n/Fs) +
13  sin(2*pi*9000.*n/Fs);
14  %% aplica filtro
15  sinal_filtrado = filter(b, a, entrada_discretizada);
16  %% calcula espectros e plota sinais
17  fft_sinal_entrada = abs(fft(entrada_discretizada)/N);
18  fft_sinal_filtrado = abs(fft(sinal_filtrado)/N);
19  f_resol = Fs/N;
20  f = n.*f_resol;
21  subplot(1,2,1); plot( entrada_discretizada); hold on; plot(sinal_filtrado,'r');
22  subplot(1,2,2); plot(f(1:N/2), fft_sinal_entrada(1:N/2)); hold on;
23  plot(f(1:N/2),fft_sinal_filtrado(1:N/2),'r');
```